习题课3

苏州大学 计算机科学与技术学院
汪笑宇
Email: xywang21@suda.edu.cn

作业3-1

- 使用动态规划算法求解最大子数组问题（问题定义参照教材p39：寻找数组$A[1..n]$中，和最大的非空连续子数组），请按照动态规划算法求解的4个步骤进行解答，并分析算法渐近时间复杂度。

作业3-1 (续)

1. 最优解结构特征：
   设m_i表示以$A[i]$结尾的最大子数组和，则
   当$m_{i-1} < 0$时，$m_i = A[i]$ ①
   当$m_{i-1} \geq 0$时，$m_i = m_{i-1} + A[i]$ ②
   即$m_i = \max\{A[i], m_{i-1} + A[i]\}$
   求解原问题时，遍历所有m_i取最大值即可

作业3-1 (续)

■ 最优子结构证明（即证明m_i包含了m_{i-1}的值， ② 成立）：
“剪切-粘贴”+ 反证法：

1. （明确反证假设）假设当以$A[i-1]$结尾的最大子数组和非负时，以$A[i]$结尾的最大子数组不包含以$A[i-1]$结尾的最大子数组。即当$m_{i-1} \geq 0$时，m_i的计算不包含m_{i-1}的值。

2. （根据反证假设给出具体替换值的假设）由于以$A[i]$结尾的最大子数组必定包含$A[i]$，现考虑以$A[i-1]$结尾的子数组。根据1的假设，当$m_{i-1} \geq 0$时，必然存在以$A[i-1]$结尾的另一个子数组和为m'_{i-1}，使得$m_i = A[i] + m'_{i-1}$。

3. （“剪切”-“粘贴”）由于$m_{i-1} \geq 0$，以$A[i]$结尾的最大子数组一定会包含以$A[i-1]$元素结尾的某个子数组。考虑将以$A[i-1]$结尾的最大子数组“剪切”并“粘贴”为更优的对应和为m'_{i-1}的子数组，即构造一个新的以$A[i]$结尾的最大子数组。

4. （导出矛盾）显然该子数组合法，且可得到$A[i] + m'_{i-1} > A[i] + m_{i-1}$，即$m'_{i-1} > m_{i-1}$，这与$m_{i-1}$是以$A[i-1]$结尾的最大子数组和矛盾。

作业3-1 (续)

2. 递归解：

- $A[1..n]$：待求解序列
- $m[i]$：以下标i元素结尾的最大子数组和
- $s[i]$：以下标i元素结尾的最大子数组的起始位置

$$m[i] = \begin{cases} A[i], & i = 1, \\ \max\{m[i-1] + A[i], A[i]\}, & 2 \le i \le n. \end{cases}$$

$$s[i] = \begin{cases} s[i-1], & m[i-1] + A[i] \ge A[i], \\ i, & m[i-1] + A[i] < A[i]. \end{cases}$$

作业3-1 (续)

3. 计算最优解的值

```
FIND_MAXIMUM_SUBARRAY_DP(A)
1   n ← A.length
2   let m[1..n] and s[1..n] be new arrays
3   m[1] ← A[1];  s[1] ← 1
4   for i ← 2 to n do
5       if m[i – 1] + A[i] ≥ A[i]
6           m[i] ← m[i – 1] + A[i]
7           s[i] ← s[i – 1]
8       else m[i] ← A[i]; s[i] ← i
9   ans ← –∞ //ans记录最大子数组和
10  for i ← 1 to n do
13      if m[i] > ans
14          ans ← m[i];  t ← i
15  return m, s and t
```

$O(n)$

作业3-1 (续)

4. 构造最优解

```
MAXIMUM_SUBARRAY _OUTPUT(m, s, t)
1  print “最大子数组和：” m[t]
2  print “数组下标范围：” s[t] “-” t
```

作业3-2

- 设计一个$O(n^2)$时间的算法，求一个n个数的序列的最长单调递增子序列。请按照动态规划算法求解的4个步骤进行解答。

作业3-2 (续)

1. 最优解结构特征：
   （回想最大子数组问题的动态规划解法）
   求以每一个元素结尾的单调递增子序列长度：
   该长度=前缀最长单调递增子序列长度（拼上该元素依旧单调递增）+1
   之后取最大值即可

作业3-2 (续)

2. 递归解：

- $A[1..n]$：待求解序列
- $c[i]$：以下标i元素结尾的最长单调递增子序列长度
- $s[i]$：以下标i元素结尾的最长单调递增子序列倒数第二个元素位置

$$c[i] = \begin{cases} \max\limits_{1 \le j \le i-1 \ \wedge \ A[j] \le A[i]} \{c[j]\} + 1, & \exists 1 \le j < i : A[j] \le A[i], \\ 1, & \forall 1 \le j < i : A[j] > A[i]. \end{cases}$$

$$s[i] = \begin{cases} j, & \exists 1 \le j < i : A[j] \le A[i] \\ & \quad \wedge \ c[j] \text{ is the maximum number in } c[1..i-1], \\ 0, & \forall 1 \le j < i : A[j] > A[i]. \end{cases}$$

作业3-2 (续)

3. 计算最优解的值

```
LIS(A)
1  n ← A.length
2  let c[1..n] and s[1..n] be new arrays
3  for i ← 1 to n do
4       c[i] ← 1,  s[i] ← 0
5  for i ← 1 to n do
6       m ← 0  //m记录前缀最长单调递增子序列长度
7       for j ← 1 to i – 1 do
8            if A[j] ≤ A[i] and c[j] > m
9                 m ← c[j],  c[i] ← c[j] + 1,  s[i] ← j
10 ans ← 0 //ans记录最长单调递增子序列长度
11 t ← 0  //t记录最长单调递增子序列最后一个元素下标
12 for i ← 1 to n do
13      if c[i] > ans
14           ans ← c[i],  t ← i
15 return ans, t, c, and s
```

$O(n^2)$

作业3-2 (续)

4. 构造最优解

```
LIS_OUTPUT(t, c, s, A)
1  print "最长单调递增子序列长度："c[t]
2  LIS_RECURSIVE(t, c, s, A)
```

```
LIS_RECURSIVE(t, c, s, A)
1  if  s[t] ≠ 0
2        LIS_RECURSIVE(s[t], c, s, A)
3  print "A["t"] = "A[t]
```

作业3-3

- 找零问题：设数组$A[1..n]$ 中的元素表示n个零钱面值，设计一个动态规划算法寻找可找开某个金额的最少零钱数量，及相对应的找零方案，若不存在找零方案则返回错误信息。例如：$A = \{1, 2, 5\}$，找零金额为11，则最少零钱数量为3，找零方案为5+5+1。请按照动态规划算法求解的4 个步骤进行解答。

作业3-3 (续)

1. 最优解结构特征：
   找开i $(0 \le i \le N)$元钱的最少零钱数量 =
   找开$(i - A[j])$元钱的最少零钱数量 + 1
2. 递归解：$m[i]$：找开i元钱所需的最少零钱数量

$$m[i] = \begin{cases} 0, & i = 0, \\ \infty, & 0 < i < \min\{A[1..n]\}, \\ \min\limits_{1 \le j \le n, i - A[j] \ge 0} \{m[i - A[j]]\} + 1, & i \ge \min\{A[1..n]\}. \end{cases}$$

作业3-3 (续)

3. 计算最优解的值

```
CHANGE(A, N)
1  let m[0..N] and s[1..N] be new arrays
2  n ← A.length,  m[0] ← 0,  p ← ∞
3  for i ← 1 to n do
4       if A[i] < p
5            p ← A[i]  //p记录最小零钱面值
6  for i ← 1 to p – 1 do
7       m[i] ← ∞
8  for i ← p to N do
9       t ← ∞  //t记录找开(i – A[j])金额的最少零钱数量
10      for j ← 1 to n do
11           if m[i – A[j]] < t
12                t ← m[i – A[j]],  s[i] ← A[j]
13      m[i] ← t + 1
14 return m and s
```

作业3-3 (续)

4. 构造最优解

```
CHANGE_OUTPUT(N, m, s)
1  if m[N] = ∞
2        print “无法找开”
3  else i ← N
4        while i > 0 do
5              print “面值”s[i]
6              i ← i – s[i]
7        print “最少零钱数：”m[N]
```

作业4-1

- 对于活动选择问题，并不是所有贪心方法都能得到最大兼容活动子集。请举例说明，在剩余兼容活动中选择持续时间最短者不能得到最大集。类似地，说明在剩余兼容活动中选择与其他剩余活动重叠最少者，以及选择最早开始者均不能得到最优解。

作业4-1 (续)

注：举例言之有理即可

- 假设存在3个活动$\{a_1, a_2, a_3\}$，其发生时间为[1,5), [4,6), [5,10)，在剩余兼容活动中选择持续时间最短者得到的兼容活动子集是$\{a_2\}$，实际最大兼容活动子集为$\{a_1, a_3\}$，在剩余兼容活动中选择持续时间最短者不能得到最优解。
- 假设存在11个活动$\{a_1, a_2, ..., a_{11}\}$，其发生时间为[1,5), [5,10), [10,15), [1,3), [3,8), [8,12), [12,15), [1,5), [10,15), [1,3), [12,15)，在剩余兼容活动中选择与其他剩余活动重叠最少者得到的兼容活动子集是$\{a_1, a_2, a_3\}$，而最大兼容活动子集$\{a_4, a_5, a_6, a_7\}$，在剩余兼容活动中选择与其他剩余活动重叠最少者不能得到最优解。
- 假设存在3个活动$\{a_1, a_2, a_3\}$，其发生时间为[1,5), [2,4), [4,8)，在剩余兼容活动中选择最早开始者得到的兼容活动子集是$\{a_1\}$，实际最大兼容活动子集为$\{a_2, a_3\}$，在剩余兼容活动中选择最早开始者不能得到最优解。

作业4-2

- 定义装箱问题为：有n个物品，编号为$1, 2, ..., n$，其中第i $(1 \le i \le n)$号物品的重量为$w_i \in (0, 1]$。需寻找一个使得n个物品全部装箱的装箱方案，且装入的箱子数量最少。注意，这里每个箱子容量都是1。

- 其中FirstFit算法是比较常用的在线装箱算法。在线算法指的是算法执行时不需要知道全局的输入信息。FirstFit算法的基本思想是：对于每个物品，装入第一个可以装进去的箱子。若前面有物品的箱子都无法装入，则新开一个箱子装入。

- (1) 请写出FirstFit算法的伪代码；(2) 请证明该算法得到的解$SOL \le 2OPT$。（提示：最多只有一个箱子是半满的，因此可以找到所有物品重量之和与(SOL-1)/2的关系；且OPT一定不小于所有物品重量之和）

作业4-2 (续)

```
BINPACKING(w)
1  n ← w.length
2  k ← 1
3  for i ← 1 to n do
4       将第i号物品放入1..k号箱子中第一个可以放入的箱子
5       if 第i号物品无法放入1..k号箱子任一个中
6            新开一个箱子放入第i号物品
7            k ← k + 1
8  return k个箱子的装箱情况
```

■近似比证明：

观察得知：最多只有一个箱子不能达到半满状态

于是有 $\sum_i w_i > (\mathrm{SOL}-1)/2$，并且有 $\mathrm{OPT} \geq \sum_i w_i$

于是 $\mathrm{SOL}-1 < 2\sum_i w_i \leq 2\mathrm{OPT}$，即 $\mathrm{SOL} < 2\mathrm{OPT}+1$

因为SOL和OPT都是正整数，于是 $\mathrm{SOL} \leq 2\mathrm{OPT}$

作业4-3

- 0-1背包问题变种：给定n个物品和一个容量为W的背包，第$i\ (1 \le i \le n)$个物品重量为w_i，应当如何选择装入背包的物品使得不超过背包容量时的总重量最大？请给出求解该问题的贪心算法基本思想及伪代码，并求解近似比(SOL/OPT)。
  - 近似算法（贪心策略）：将物品按照重量从大到小排序依次装入，直到无法装入为止。该方法近似比为1/2

作业4-3 (续)

```
KNAPSACK1_APPROX(w, W)
1  将w[1..n]单调递减排序
2  n ← w.length,  sum ← 0
3  A ← ∅
4  for i ← 1 to n do
5        if sum + w[i] ≤ W
6              A ← A ∪ {w[i]}
7              sum ← sum + w[i]
8  return A and sum
```

■近似比证明：
首先受容量限制一定有
$OPT \le W$

- 可装入的物品分两种情况：
  (1) 最重物品重量超过$W/2$；
  (2) 最重物品重量不超过$W/2$。
- 第(1)种情况：最重物品重量超过$W/2$时，因为按照重量从重到轻放入背包，则最重物品被装入，此时的解$SOL \ge W/2 \ge OPT/2$，即$SOL/OPT \ge 1/2$。
- 第(2)种情况：假设前i个物品都可装入背包，若此时装包重量依旧不超过$W/2$，则$i+1$号物品依旧可装入，直到装包重量超过$W/2$时，后续物品无法判断是否能装入。此时仍有$SOL \ge W/2 \ge OPT/2$，即$SOL/OPT \ge 1/2$。

综上，算法近似比为1/2。